CheepSync: A Time Synchronization Service for Resource Constrained Bluetooth LE Advertisers

Sabarish Sridhar, Prasant Misra, Gurinder Singh Gill, and Jay Warrior

This article presents and describes *Cheep*Sync, a time synchronization service for BLE advertisers, especially tailored for applications requiring high time precision on resource constrained BLE platforms. Designed on top of the existing Bluetooth v4.0 standard, the *Cheep*Sync framework utilizes low-level timestamping and comprehensive error compensation mechanisms for overcoming uncertainties in message transmission, clock drift, and other system-specific constraints.

Abstract

Clock synchronization is highly desirable in distributed systems, including many applications in the Internet of Things and Humans. It improves the efficiency, modularity, and scalability of the system, and optimizes use of event triggers. For IoTH, BLE — a subset of the recent Bluetooth v4.0 stack — provides a low-power and loosely coupled mechanism for sensor data collection with ubiquitous units (e.g., smartphones and tablets) carried by humans. This fundamental design paradigm of BLE is enabled by a range of broadcast advertising modes. While its operational benefits are numerous, the lack of a common time reference in the broadcast mode of BLE has been a fundamental limitation. This article presents and describes *Cheep*Sync, a time synchronization service for BLE advertisers, especially tailored for applications requiring high time precision on resource constrained BLE platforms. Designed on top of the existing Bluetooth v4.0 standard, the *Cheep*Sync framework utilizes low-level timestamping and comprehensive error compensation mechanisms for overcoming uncertainties in message transmission, clock drift, and other system-specific constraints. *Cheep*Sync was implemented on custom designed nRF24Cheep beacon platforms (as broadcasters) and commercial off-the-shelf Android ported smartphones (as passive listeners). We demonstrate the efficacy of *Cheep*Sync by numerous empirical evaluations in a variety of experimental setups, and show that its average (single-hop) time synchronization accuracy is in the 10 μs range.

Introduction

A common time reference is an important requirement for distributed systems [1], and developing such a service on *constrained* devices is particularly challenging [2–10]. In the recent past, a new class of constrained platforms based on Bluetooth Low Energy (BLE)[1] have emerged. BLE is different from other wireless technologies because it combines a standardized communication technology designed for low-power systems, and a new sensor-based data collection framework. It also offers easy integration with most handheld devices (e.g., smartphones and tablets), something toward which traditional wireless sensor networks (WSNs) are still working. BLE has inherited several technical features from classic Bluetooth that provide for robust reliable connections. However, the most significant difference is its asymmetric design. While the communication foundation is based on a master-slave architecture, it offers a new feature in the form of an *advertisement* (configurable through the broadcast/peripheral mode of BLE). This new mode offers unidirectional correspondence between two or more LE devices using advertising events, thereby achieving a communication solution without entering into a bonded connection (as required by classic Bluetooth devices). Such a loosely coupled manner of data transfer is undoubtedly more energy efficient, but also unearths other limitations. For example, the broadcast mode of communication *does not* provision for a time synchronization service; even though this feature is included in the solution stack, it can only be availed upon pairing.

Challenges

BLE provides a range of broadcast advertising modes, of which the *most* energy efficient is the non-connectable undirected advertising mode (`ADV_NONCONN_IND`): a transmit-only broadcaster mode without any listen window. Establishing time synchronization in this mode of BLE operation is challenging due to many reasons. First, the traditional techniques of message passing among different elements is not supported by this network architecture (i.e., Bluetooth v4.0), and as a result, timing uncertainties cannot be compensated by exchanging time stamped packets, or "pings" between nodes. Second, devices receiving advertisement data (e.g., smartphones) have high functional asymmetry compared to the BLE broadcast units. They typically run on a multithreaded and multitasking operating system (OS) (e.g., Android) where the measure of system latencies and their associated uncertainties can be

[1] BLE Specification Adopted Documents. https://www.bluetooth.org/en-us/specification/adopted-specifications

Sabarish Sridhar is with M.S. Ramaiah Institute of Technology, Bangalore; Prasant Misra and Gurinder Singh are with the Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bangalore; Jay Warrior is with Mobiatrics LLC. Sabarish Sridhar was an intern and Jay Warrior was the chief technologist at the Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bangalore during the course of this work.

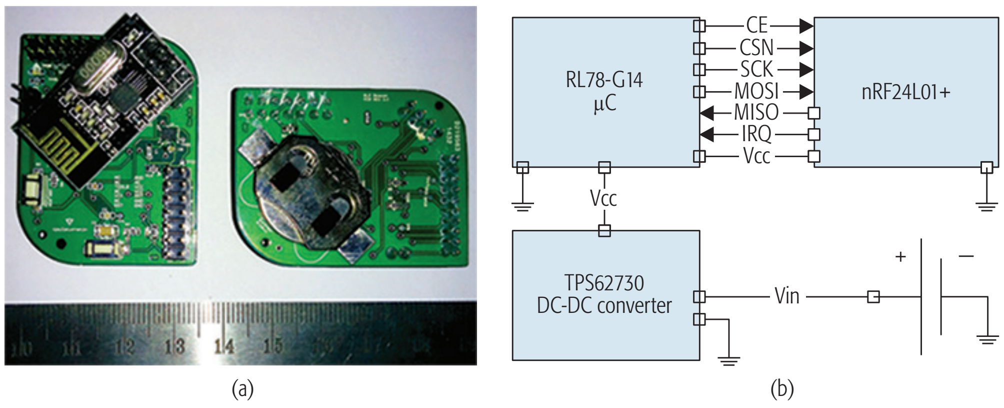

Figure 1. nRF24*Cheep*: custom designed BLE Beacon platform. The major components include: *RL78-G14* microcontroller, *NRF24L01+ 2.4 GHz* RF transceiver, *TPS62730* synchronous step-down DC-DC converter, and *CR2032* battery holder. a) platform overview; b) functional representation of the major platform components.

The system is composed of two units: beacon and control. The beacon unit consists of resource constrained sensing tags that are deployed in the region of interest. The control unit consists of a resourceful gateway device capable of listening and receiving broadcast data contained in BLE advertisements.

many orders of magnitude higher than those at the transmitter end. Third, low-level timestamping on such multifunctional receiver devices can be performed to a certain limit and is subject to system restrictions of the underlying firmware. Therefore, motivated by the need to overcome the above limitations but be able to establish a common time reference across resourced constrained BLE devices operating in the `ADV_NONCONN_IND` mode, we propose *Cheep*Sync.

Contributions and Road Map

Due to the architectural constraints, the key ideas of *Cheep*Sync are:

- Not to synchronize the nodes in the network, but to make the devices that use these nodes to synchronize
- Piggyback on the device mobility aspect (as they are inevitably carried by people) and use them as "synchronization mules" for the "broadcast" system

Thus, it offers a flexible piggyback design wherein running the time service does not require data transactions to be temporarily suspended. Therefore, time synchronization with *Cheep*Sync is highly implicit rather than explicit. Since *Cheep*Sync rides on the BLE broadcast framework, it is scalable to the point that the framework has to offer.

In this article, we describe our experiences in building a custom BLE beacon platform that uses BLE fakery over a general-purpose radio — a tool for conducting research in this direction — in the next section. It is followed by a detailed design and analysis of the *Cheep*Sync architecture and performance that is able to achieve an average time synchronization accuracy in the range of 10 µs. The final sections provide a concise background of existing work in this related field, and we conclude with a summary of the areas covered in the article.

System Overview

The system is composed of two units: beacon and control. The *beacon unit* consists of resource constrained sensing tags that are deployed in the region of interest. They are responsible for measurement of simple physical parameters, and disseminating that information through undirected BLE broadcasts. The *control unit* consists of a resourceful gateway device capable of listening and receiving broadcast data contained in BLE advertisements. In our case, the beacon unit is a custom designed nRF24*Cheep* BLE platform, and the control unit is an Android v4.4.4 smartphone that uses Bluedroid (the default Android Bluetooth stack).

nRF24*Cheep*

The custom designed beacon platform, nRF24*Cheep* (Fig. 1), consists of an RL78-G14 microcontroller, an nRF24L01+ 2.4 GHz RF transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), a TPS62730 synchronous step-down DC-DC converter, and a CR2032 battery holder. The microcontroller has an RL78 core with 16–512 kB flash memory, and operates at a maximum clock speed of 64 MHz with a high precision (±1 percent) on-chip oscillator. It also provides a range of *interval timers* for different application requirements. The nRF24L01+, although not strictly compatible with BLE, can be made to operate as a BLE transmitter by configuring the transceiver settings.[2] This allows BLE listeners/scanners to view the nRF24L01+ as a BLE device and decode its advertisement data. The nRF24L01+ interfaces with the application controller (RL78-G14) over a high-speed Serial Peripheral Interface (SPI) bus. Enhanced ShockBurst™, designed to handle all the high-speed link layer operations, is based on packet communication and supports 1 to 32 bytes of dynamic payload length. Data flow between the radio front-end and the microcontroller is through internal first-in first-out buffers (FIFOs).

The nRF24L01+ module has the following eight interfacing pins, of which four are SPI related: CSN, SCK, MISO, MOSI; the remaining ones are Vcc, GND, IRQ, and CE. CE is used to control data transmission and reception in TX and RX modes, respectively. In TX mode, CE is always kept low except when the packet has to be transmitted, and is done by loading the TX FIFO and then toggling the CE pin. IRQ is the interrupt pin, and can be used to assert three internal interrupts: data received, data transmitted, and maximum number of transmit retries reached.

[2] The necessary RF configurations for BLE compliance can be obtained from Bit-Banging Bluetooth Low Energy: http://goo.gl/JDpflR

™ Enhanced ShockBurst is a trademark of Nordic Semiconductor.

Send time is the time spent by the transmitter to assemble the message and trigger the send request to the MAC layer. It is, therefore, a function of the processor load and the system call overhead of the respective OS ported on the transmitter platform.

*Cheep*Sync: Time Synchronization Protocol

The basic *Cheep*Sync mechanism uses the broadcaster mode of BLE to transmit a single advertisement packet from the beacon (transmitter) unit to the control (receiver) unit. The broadcasted message contains:

- The transmitter's current timestamp value, which is a counter field that increments every *interval* millisecond, and is the estimated local time at the transmission of the advertisement packet;
- The aggregate delay incurred during the transmission of the previous packet

On message reception, the receiver obtains the corresponding "wall'" clock time expressing time (in nanoseconds) since the epoch. In principle, two broadcast packets provide a *synchronization* point between the transmitter and the receiver.[3] The difference between the local and "wall" clock time of a synchronization point estimates the clock offset of the transmitter.

The counter value *ts_counter* is needed to estimate the time elapsed on the beacon unit. If the underlying platform uses a b bit field to store the timestamp value, and the timer fires every *interval* ms, the total time that can be represented by the timestamp field is ($2^b * interval$) ms. Therefore, the configuration of $b = 24$ bits and $interval = 100$ ms can be used to make the timestamp value work for approximately 19 days without rolling over.[4]

In the next subsection, we discuss the general uncertainties associated with RF message delivery and then converge to our specific use case. It is then followed by a detailed explanation of the structure of the BLE advertisement packet with specifics into the packet restructuring as per the nRF24L01+ Enhanced ShockBurst protocol engine.

Sources of Time Synchronization Error

We shall use the following error decomposition model [2–4, 9, 11] to better understand the sources of latency, and modify it according to the specifics of the platform and radio of interest.

Send Time: It is the time spent by the transmitter to assemble the message and trigger the send request to the medium access control (MAC) layer. Therefore, it is a function of the processor load and the system call overhead of the respective OS ported on the transmitter platform. nRF-24*Cheep* does not have an OS port, and makes direct system calls to the underlying hardware without any (potential) soft routing. This enables more user control over different system modules (albeit with increased complexity), and thus helps to reduce delays that are typically nondeterministic and were previously difficult to calibrate.

Access Time: It is the delay incurred waiting for access to the transmit channel up to the point when transmission begins, and is specific to the MAC protocol in use. It is considered the least deterministic part of the message delivery system. BLE *does* provision for a (time/frequency-division multiple access, TDMA/FDMA) MAC, but it is only operational in connection mode. Access control rules have not been defined for the BLE broadcast mode of communication, so packets get pushed on to the physical channel as and when they are flagged for transmission.

Transmission Time: A function of the length of the message and the radio speed, it is the time taken by the transmitter to transmit the message and is a deterministic component.

Propagation Time: Once the message has left the transmitter, it is the time needed to transit to the receiver. For many application requirements (wherein the channel length is under 300 m), this delay is highly deterministic and less than 1 μs.

Reception Time: It is the time taken by the receiver to receive the message. In our case, it is the most nondeterministic part of the message delivery mechanism as the receivers are Android ported smartphones that run multiple tasks and process threads at the same time.

Receive Time: It is the time required to process and notify the incoming message to the receiver application.

We perform low-level timestamping, at both the transmitter and receiver ends, to overcome the above stated uncertainties in a message transaction, the details of which are discussed subsequent to the BLE advertisement packet format that follows below.

Advertisement Packet Format

There is a single format for a BLE (advertisement or data) `Packet`, and it consists of the following four fields:

1. Preamble (1 octet)
2. Access address (4 octets)
3. Protocol data unit (PDU: conventionally[5] 2–39 octets, but limited to 2–32 octets in nRF Enhanced ShockBurst™ packet format)
4. Cyclic redundancy check (CRC, 3 octets)

As per the core specifications of an advertisement packet, the 8-bit preamble and 32-bit access address were set to `10101010b` and `0x8E89BED6`, respectively. The preamble is used in the receiver to perform frequency synchronization, symbol timing estimation, and automatic gain control training. A 24-bit CRC is appended to the end of every packet, and is calculated over the PDU. It is important to note that some fields in the packet definition, marked RFU, are reserved for future use, and are set to zero at transmission and ignored upon receipt. Depending on the PDU size, a BLE advertisement packet length could vary from 10 to 40 octets in the nRF Enhanced ShockBurst mode (as opposed to the standard 10–47 octets payload). The *advertisement* channel PDU has a 16-bit header and a variable-size payload.

Header: The header consists of the following six fields spanning over 2 octets:

1 `PDU type` (4 bits);
2 `RFU` (2 bits);
3 `TxAdd` (1 bit);
4 `RxAdd` (1 bit);
5 `Length` (6 bit);
6 `RFU` (2 bit).

The `PDU type` was set to `ADV_NONCONN_IND (0010b)` for *transmitting* non-connectable undirected advertising events. The following `RFU`, `TxAdd`, and `RxAdd` fields were not used, and hence were set to zero. The payload size is indicated by the `Length` field, and can vary between 6 to 30 octets (instead of the standard 37 octets).

Payload: The `Payload` for the `ADV_NON-`

[3] We explain how the determinism of time delays on the nRF24*Cheep* beacon platform benefits our implementation. Therefore, for our platform of choice, only a single broadcast packet (instead of two) is apt for reliably synchronizing the transmitter and the receiver below.

[4] Our system design is based on the assumption that there must be a control unit that passes by the beacon units at least once over a 19-day period. However, under overriding circumstances, the system can determine the number of counter rollovers.

[5] By conventional/standard, we mean the guidelines provided in the Bluetooth core specification v4.

CONN_IND PDU consists of the following two fields:
1. AdvA (6 octets)
2. AdvData (0–24 octets, instead of the standard 0–31 octets)

The AdvA field holds the device address of the advertiser, which can be either public (if TxAdd = 0) or random (if TxAdd = 1). The AdvData field contains the advertisement data, and it consists of two logical parts: significant and non-significant. The significant part contains a sequence of AD structures. Each AD structure consists of the following two fields to populate a separate item of user data:
1 Length (1 octet)
2 Data (Length octets)
- AD type (1 octet)
- AD data (Length - 1 octets)

The non-significant part extends the advertising data to the remaining octets and contains all zeroes. For our implementation, we use the AD structures (defined in the core specification) presented in Table 1.

The AD type Flags sets the discoverability preference of the device, and the General Discoverable Mode makes it detectable unconditionally. The Local Name (Shortened) AD type sets the short user-readable name of the device. The Manufacturer Specific Data AD Type is a generic, freely formattable data field, and includes the 24-bit timestamp value and an 8-bit transmit time delay (of the previous packet).

Timestamping at the Transmitter

On the nRF24*Cheep* platform, an advertisement packet is transmitted by loading the TX FIFO and pulling the CE pin to a high state. One of the fields that is pushed into the SPI buffer is the current timestamp value. A second timestamp is also recorded once the TX_DS interrupt is seen by the microcontroller (i.e., when the radio's IRQ pin is pulled down low), signaling the success of the transmit event. The difference between these two timestamps provides an accurate estimate of the time delays incurred due to send, access, and transmission; and this information is encapsulated into the next advertisement packet. As discussed above, due to the high determinism of send, access, and transmission time delays on the nRF24*Cheep* beacon platform, our implementation uses a *single* broadcast packet (instead of two) to reliably synchronize the transmitter and the receiver. The timestamping characteristics at the transmitter end are shown in Fig. 2, depicted as a histogram showing the distribution of the transmitting time interval recorded for 35,000 broadcast packets. The distribution appears Gaussian with best fit parameters of $\mu = 0.201829$ μs and a minuscule $\sigma^2 = 5.19537e{-}07$ μs. This latency characterization supports the determinism of our approach on the transmitter end.

Timestamping at the Receiver

The reception time on the Android phone can be further divided into the following delivery delays:
- Time taken by the Broadcom BCM radio to receive the message and raise an interrupt
- Time taken by the standard UART platform driver to hook into the BCM radio and register a RX event

Field	Value
Length AD type AD data	2 octets Flags General discoverable mode
Length AD type AD data	6 octets Local name (shortened) –
Length AD type AD data	4 octets Manufacturer-specific data Timestamp, transmit time delay (previous packet)

Table 1. Utilized AD types.

- Time taken by the Bluedroid stack to poll the UART driver, waiting to check if there are any bytes to be read (using the userial_read_thread() method in userial.c)

Therefore, there is a significant nondeterministic time lag between the instants when an RF message is actually received by the radio to when it is processed by the Bluedroid stack. Moving to software layers beyond the UART will limit portability across different Android phones. Taking these facts into consideration, the lowest layer accessible entity on the Android Bluedroid stack is userial.c.[6]

userial.c interfaces with the standard UART driver, and provides a common interface for the time-sync to work on multiple phones without introducing many hardware changes. However, timestamping at the level of userial.c introduces two complications. First, it is a generic container for catching all types of events (notification and other messages) sent out by the BCM radio to the underlying driver; hence, there is a need to correctly identify the Bluetooth event. Second, this problem can be overcome by waiting for userial.c to process the event list, but would introduce finite variable delays. Therefore, timestamps are recorded at the instant when an event is received at userial.c, but before any processing is performed on the list.

The clock_gettime(CLOCK_REALTIME) method, which returns the real-time clock of the system in nanoseconds since the Epoch (00:00 1 January, 1970 UTC), is utilized to perform low-level timestamping on the receiver (phone) end. This representative value is passed on to the higher-layer application, and the receive timestamp corresponding to the received BLE packet is subsequently determined. It is possible that a notification was received by userial.c from the radio, and the application layer was called following that particular notification being recorded as a timestamp. This introduces an error since the timestamp of the notification and not that of the message is being used by the program.

To overcome this problem, we adopt an algorithmic approach that determines the timestamp corresponding to a received packet. The algorithm first iteratively records the timestamps for the previous and current packets, and then subtracts each current packet's timestamp value from every previous packet's timestamp value. Upon completion, the value with the least devi-

The algorithm first iteratively records the timestamps for the previous and current packets, and then subtracts each current packet's timestamp value from every previous packet's timestamp value. Upon completion, the value with the least deviation is taken as the best fit timestamp for the particular packet.

[6] In the Android Bluedroid stack, userial.c is located at: /external/bluetooth/bluedroid/hci/src/

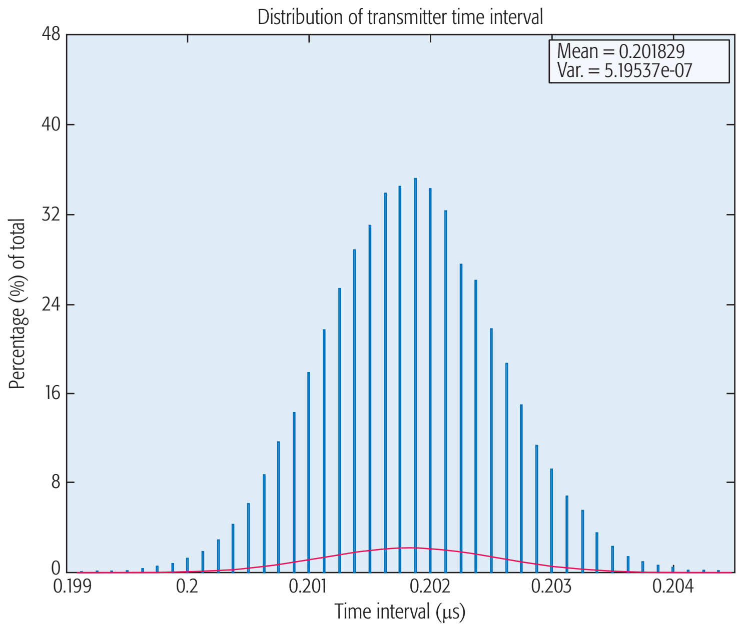

Figure 2. Transmitter side timestamping characteristics. Histogram showing the distribution of transmitter time interval recorded for 35,000 broadcast packets, grouped into 1 µs buckets. The curve is a plot of the best fit Gaussian parameters with $\mu = 0.201828$ µs and $\sigma^2 = 5.19537e{-}07$ µs.

ation is taken as the best fit timestamp for the particular packet.

Clock Drift Management

The clock quality and speed on the transmitter (nRF24*Cheep*) and the receiver (Nexus~5 phone) are vastly different. *Cheep*Sync makes continuous skew adjustments over a measurement window on the phone unit as: $k = (o_i - \overline{o})/(t_i^r - \overline{t^r})$; where, $\overline{t^r}$ is the average elapsed time at the reference clock and $\overline{o}$ is the average offset up to the i^{th} sample point. This linear regression method offers a fast mechanism to find the frequency and phase errors over time.

Figure 3 shows the performance of time synchronization with and without clock drift compensation for an interval value of 100 ms over a period of 13 h. Without drift compensation, the mean error in synchronizing the transmitter and receiver clock is in the millisecond range, but can be brought down to a few microseconds with correction (including both low-level timestamping and clock drift management). For the specific case of *interval* = 100 ms, the error before drift correction was $\mu = 2.62$ ms, $\sigma^2 = 6.37312$ ms, and 95 percent cumulative probability of 8.2 ms; while the respective estimates after correction were recorded as $\mu = 0.0667$ ms, $s^2 = 3.89512e{-}02$ µs, and 0.64 µs at the 95 percent cumulative probability level. We also experimented with an interval value of 200 µs [12], but the interval timer of 100 ms was chosen due to its high stability and lower energy cost.

Multi-Device Time Synchronization

Considering the system dynamics, there are two forms of time synchronization across multiple devices.

Many-Tx-to-One-Rx Synchronization: The scenario of synchronizing multiple Tx's to a single Rx is a simple extension of the case of deriving an estimate of time for a single (transmitter, receiver) pair, wherein the control unit becomes the reference point to perform synchronization. A reference point contains a pair of local and global timestamps where both of them refer to the same time instant. The control unit receives periodic broadcasts from beacon units within their coverage zone; otherwise, their records are not entered. When the control unit collects the required measurement points, it estimates the skew and offset of the observed beacons and derives their coordinated time measure with respect to the global time.

Many-Tx-to-Many-Rx Synchronization: The scenario of synchronizing multiple Tx's to multiple Rx's combines the above system infrastructure with a mechanism for different control units to share the skew and offset information of already visited beacon units. This could be achieved by peer-to-peer interaction between the control units or through the cloud infrastructure. For ease of implementation, we choose to take the latter alternative with the Google Cloud platform. In this case, the absolute timestamp of the control unit along with the timestamp of the beacon unit is inserted into the cloud. This data is then made accessible to other control units using the Google App Engine database. Whenever the timestamp of a new beacon unit is recorded by a control unit, a notification is sent to all other control units using Google cloud messaging. Once a control unit receives this message, it observes the last time it obtained the timestamp of the same beacon and computes the time difference. For instance, let the first control unit record timestamp t_a from beacon i at time instance τ_a. Let the second control unit record another timestamp from the same beacon i with value t_b at time instance τ_b. Using this information, the offset (i.e., time difference) between the phones is measured as $(\tau_a - \tau_b - (t_b - t_a) * k)$.

The relative time on the control units (which are Android phones) may vary from one to another by up to several seconds. It is therefore necessary to constantly synchronize with the nearest Network Time Protocol (NTP) server repeatedly in order to maintain very high precision for our time synchronization. For our implementation, we used an Android NTP application,[7] and was configured to synchronize the control units at an interval of 30 s.

Porting CheepSync into Mobile Applications

The reference design of nRF24*Cheep*, source code of *Cheep*Sync, and build instructions for a custom Android ROM are publicly available at https://github.com/prasantmisra/cheep-sync, with some preliminary results reported in [13]. The accuracy levels reported in the following section can only be obtained with the nRF24*Cheep* platform, while other BLE platforms will/may lead to higher time sync error values. As for the mobile phone platform, a custom Android ROM that includes the modified `userial.c` file needs to be installed. The same link also contains an example to capture the steps for incorporating *Cheep*Sync into a mobile application.

Experimental Evaluation

In this section, we evaluate the accuracy of *Cheep*Sync in a variety of controlled and uncontrolled experiments.

*Cheep*Sync in Controlled Experiments

Study 1: The aim of this study was to obtain the baseline performance level of *Cheep*Sync for a *many-Tx-to-one-Rx* synchronization scenario. Thus, it was designed with a *static* setup of 8 beacon units and 1 control unit that was always covered by all the beacons. The beacons were configured to broadcast advertisement packets at an interval of 100 ms and at their lowest transmit power of –20 dB. This experiment was conducted for about 13 hours wherein an average of 10000 packets were received by the control unit from each beacon.

The experimentation results are shown in Fig. 4. The system-level performance of *Cheep*Sync is depicted in Fig. 4a, which shows an average error of 8 µs and a 95 percent error probability of less than 0.04 ms. Figure 4b disaggregates the combined (drift compensated) measurements into respective representations for each of the eight beacons in the system. Here, it is evident that a large percentage of the beacon units show consistent behavior with an average error level of approximately 10 µs and a worst case value of 0.04 ms; except beacon 7, which shows 2 times better average performance but shoots over in the worst case performance level.

Study 2: The aim of this study was to obtain the baseline performance level of *Cheep*Sync for the scenarios of *one-tx-to-many-rx* and *many-tx-to-many-rx* synchronization. For the prior scenario, the static setup consisted of a single beacon and two control units, while for the latter scenario, the setup consisted of eight static beacons and two static control units. The control units were always kept within the coverage range of the beacon(s), while all other beacon/control unit configuration parameters were kept consistent with study 1. Figures 4c and 4d show the cumulative time-sync error (i.e., the time difference between control units as explained earlier) distributions for the prior and latter setup, respectively, and it is conditioned on contributing error factors including clock drift, NTP drift, and so on. The result obtained from Fig. 4c suggests that there is a 95 percent probability of the time synchronization error to be less than 5 ms with mean < 10 ms. A similar observation is also noted from Fig. 4d where the result suggests that there is a 90 percent probability of the time sync error to be less than 10 µs, although with the possibility of errors as large as 10 ms for 5 percent higher confidence levels.

*Cheep*Sync in Uncontrolled Experiments

Study 3 — This experiment was designed to evaluate the performance of *Cheep*Sync in an uncontrolled setup, and hence quantify its deviation from the respective benchmark results obtained from studies 1 and 2. The experimentation space was in a [20 × 20] m portion of our office floor. For our evaluation, we took the services of two people who were each handed an Android smartphone running the *Cheep*Sync time service and kept it for about an hour. The respective office space was instrumented with a set of five beacon

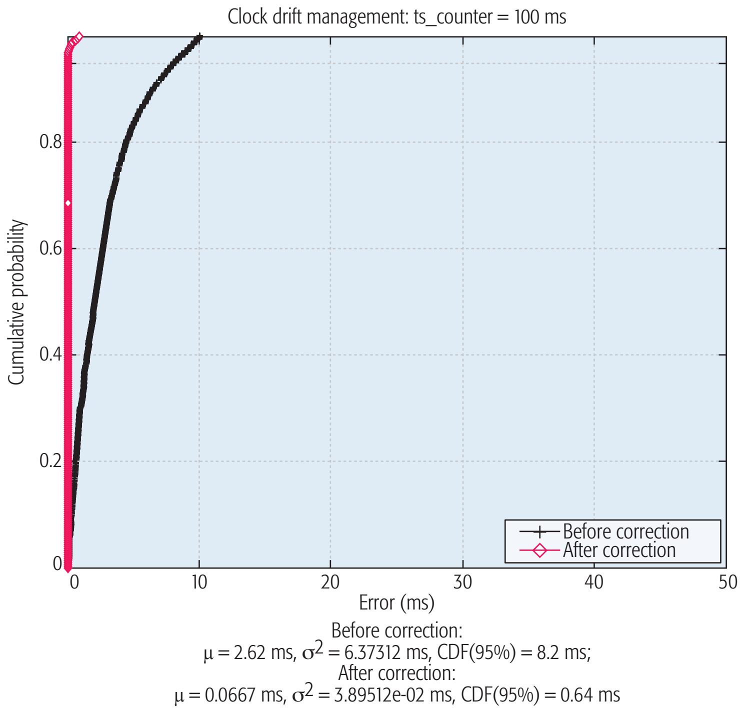

Figure 3. Clock drift management. There is one order of improvement in synchronization accuracy after compensating for clock drift, at both the average and 95 percent probability levels.

units in a manner that all beacons did not cover every end of the experiment zone. All other system configurations, such as the beaconing rate and transmit power levels, were kept the same as in study 1/2.

For the *many-tx-to-one-rx* scenario (Fig. 4e), a mean error of 12 µs was observed, while its 95 percent error probability was less than 0.04 ms. As for the *many-tx-to-many-rx* scenario (Fig. 4f), the 95 percent error probability was less than 22 ms with a mean error of 10 µs. Both of these results are in good agreement with the baseline observations recorded in Figs. 4a and 4c, thereby establishing the efficacy of *Cheep*Sync.

Discussion

Time synchronization solutions have some basic features in common: a messaging protocol to exchange timestamped packets between nodes (some acting as clients and others as time servers) in a network, techniques for overcoming nondeterministic delays, and an adjustment mechanism to update the local clock. However, they differ in aspects including whether the physical clocks of the network are kept consistent internally or are synchronized to external standards; if the server is an arbiter of the client clock or is considered as a canonical clock; and so on.

For synchronization in WSNs, there are multiple algorithms such as Reference Broadcast Synchronization (RBS) [3], Flooding Time Synchronization Protocol (FTSP) [2], Timesync Protocol for Sensor Networks (TPSN) [4], Glossy [5], and hardware assisted clock synchronization (HACS) [10] that use message passing as the basic mechanism to compensate for delays. *Cheep*Sync explores a form of synchronization that differs from traditional WSNs, and is specifical-

[7] The Android NTP application is available at https://play.google.com/store/apps/details?id=ru.org.amip.ClockSync&hl=en. While alternatives such as GPS exist, NTP was chosen due to its wide availability and benefits of scalability, self-configuration, robustness to network failures, and energy efficiency.

CheepSync explores a form of synchronization that differs from traditional WSNs, and is specifically designed for the technology segment that uses BLE undirected broadcasts and smartphones. Its fundamental design philosophy is to provision for synchronization between transmitters and receivers, as opposed to traditional WSN protocols.

ly designed for the technology segment that uses BLE undirected broadcasts and smartphones. Its fundamental design philosophy is to provision for synchronization between transmitters and receivers, as opposed to traditional WSN protocols that synchronize a set of receivers with one another. Performing receiver-receiver synchronization removes uncertainties associated with send and access delays, the biggest contributors to non-determinism in latency, from the critical time path. System-level complexities are further reduced by using symmetric WSN platforms that run on simple OSs. All of these factors conglomerate toward higher levels of synchronization accuracy than is typically evident in traditional WSNs. *Cheep*Sync, in contrast, operates on highly asymmetric devices with vast differences in system complexity. Its synchronization methodology, system architecture, and wireless communication standard utilized for delivering the respective solutions are completely different. *Cheep*Sync achieves a high synchronization accuracy that is better than the time-sync levels of RBS, TPSN, and HACS, but is less precise compared to FTSP or Glossy (Table 2). We refer

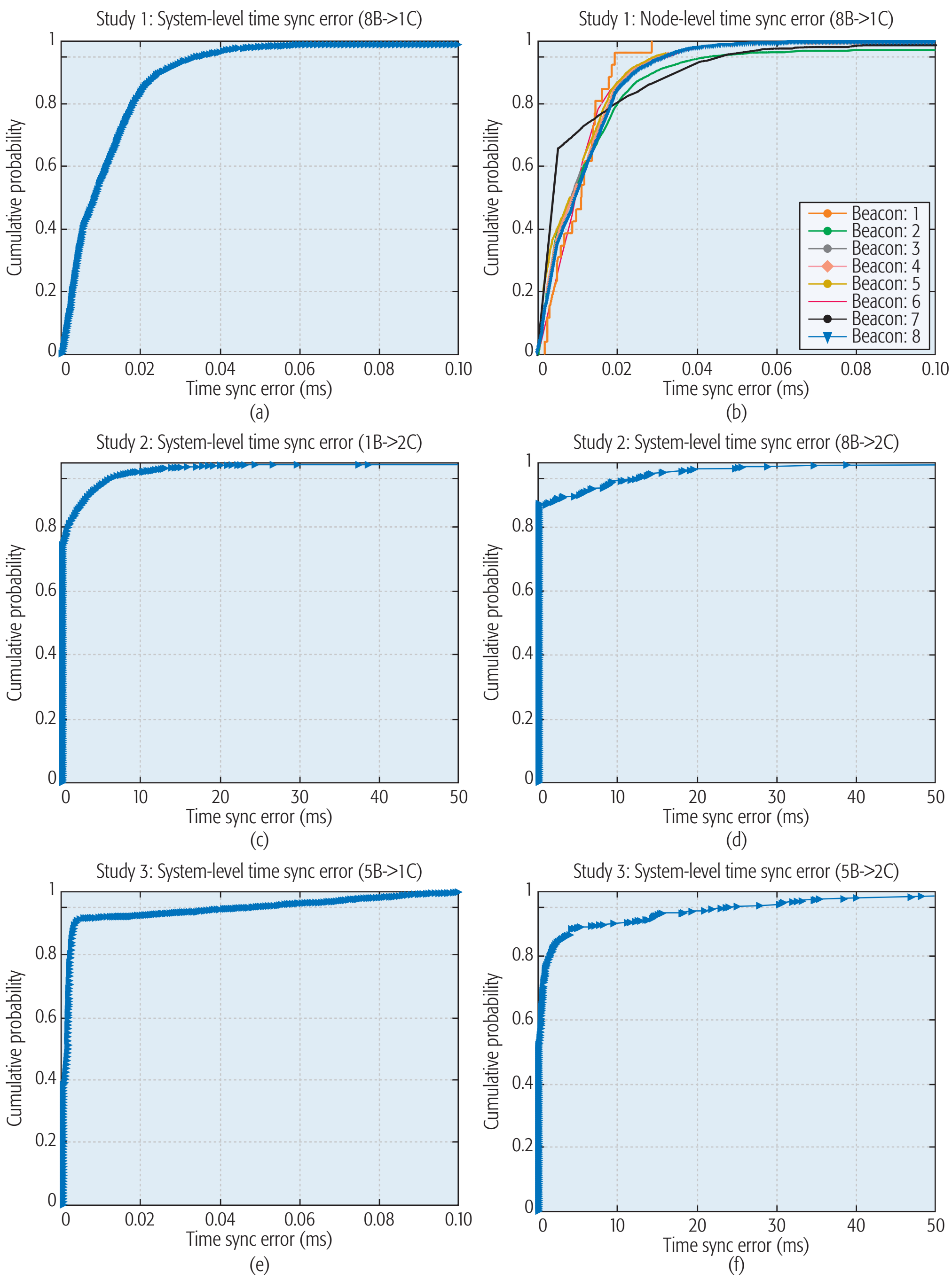

Figure 4. Performance of *Cheep*Sync. a) many-tx-to-one-rx: μ = 8 μs, CDF(95 percent) = 0.04 ms; b) many-tx-to-one-rx; c) one-tx-to-many-rx scenario: μ = 10 μs, CDF(95 percent) = 5 ms; d) many-tx-to-many-rx scenario: μ = 10 μs, CDF(95 percent) = 10 ms; e) many-tx-to-one-rx scenario: μ = 12 μs, CDF(95 percent) = 0.04 ms; f) many-tx-to-many-rx scenario: μ = 10 μs, CDF(95 percent) = 22 ms.

our reader to Serpedin *et al.* [14] for an extensive survey and analysis of various solutions in this space.

Application

The loosely coupled BLE data collection framework enables new ways of architecting Internet of Things and Humans (IoTH) systems. An example application is to monitor the *hand sanitization* status of healthcare personnel (HCP) in an intensive care unit (ICU) to control and prevent the spread of hospital acquired infections (HAIs), where unclean hands are the most common factor contributing to this cause. CleanHands [15] is a recently proposed system that uses a combination of BLE beacon tags and HCP's mobile phones for detecting occurrences of noncompliance with hand hygiene. This application requires different levels of time accuracy. For instance, a few milliseconds may be sufficient to disambiguate the handwashing event of HCPs, but a significantly higher accuracy level of a few microseconds would be required to secure the same system against possible gaming.

Conclusion

*Cheep*Sync is a time synchronization service for BLE advertisers, and therefore is a key enabler for a variety of IoTH applications where mobile crowdsourcing, using existing infrastructure and ubiquitously used platforms along with humans as the data mules, has emerged as an alternate architecture. These applications typically require different levels of time accuracy. By empirical evaluations, we show that CheepSync is capable of gracefully handling timing requirements as low as 10 μs. It is built on the existing Bluetooth v4.0 standard, and hence is generic to all devices using the low energy profile of Bluetooth.

Acknowledgments

This work has received funding from the Robert Bosch Centre for Cyber Physical Systems, IISc Bangalore under grant agreement RBCCPS/PC-0043.

Time sync. protocol	Avg. accuracy (μs) (single hop)
RBS [3]	29.10
TPSN [4]	16.90
FTSP [2]	01.48
Glossy [5]	0.50
HACS [10]	1000.00
*Cheep*Sync	10.00

Table 2. Accuracy measure of time synchronization algorithms.

By empirical evaluations, we showed that CheepSync is capable of gracefully handling timing requirements as low as 10 μs. It is built on the existing Bluetooth v4.0 standard, and hence is, generic to all devices using the low energy profile of Bluetooth.

References

[1] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Commun. ACM*, vol. 21, no. 7, July 1978, pp. 558–65.
[2] M. Maróti *et al.*, "The Flooding Time Synchronization Protocol," *Proc. 2nd Int'l. Conf. Embedded Networked Sensor Systems*, New York, NY, 2004, pp. 39–49.
[3] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *Proc. 5th Symp. Operating Systems Design and Implementation*, New York, NY, 2002, pp. 147–63.
[4] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proc. 1st Int'l. Conf. Embedded Networked Sensor Systems*, New York, NY, 2003, pp. 138–49.
[5] F. Ferrari *et al.*, "Efficient Network Flooding and Time Synchronization with Glossy," *Proc. 10th Int'l. Conf. Information Processing in Sensor Networks*, Apr. 2011, pp. 73–84.
[6] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Trans. Commun.*, vol. 39, no. 10, Oct 1991, pp. 1482–93.
[7] R. Gusella and S. Zatti, "The Accuracy of the Clock Synchronization Achieved by Tempo In Berkeley Unix 4.3bsd," *IEEE Trans. Software Engineering*, vol. 15, no. 7, Jul 1989, pp. 847–53.
[8] F. Cristian, "Probabilistic Clock Synchronization," *Distributed Computing*, vol. 3, no. 3, 1989, pp. 146–58.
[9] H. Kopetz and W. Ochsenreiter, "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. 36, no. 8, Aug. 1987, pp. 933–40.
[10] A. Rowe, V. Gupta, and R. Rajkumar, "Low-Power Clock Synchronization Using Electromagnetic Energy Radiating from AC Power Lines," *Proc. 7th ACM Conf. Embedded Networked Sensor Systems*, New York, NY, 2009, pp. 211–24
[11] M. Horauer *et al.*, "Psynutcevaluation of a High Precision Time Synchronization Prototype System for Ethernet LANs," *Proc. 34th Annual Precise Time and Time Interval Meeting*, 2002, pp. 263–79.
[12] S. Sridhar *et al.*, "CheepSync: A Time Synchronization Service for Resource Constrained Bluetooth Low Energy Advertisers," tech. rep., 2015.
[13] S. Sridhar, P. Misra, and J. Warrior, Poster Abstract: "Cheepsync: A Time Synchronization Service for Resource Constrained Bluetooth Low Energy Advertisers," *Proc. 14th Int'l. Conf. Information Processing in Sensor Networks, IPSN '15*, New York, NY, 2015, pp. 364–65.
[14] E. Serpedin and Q. M. Chaudhari, *Synchronization in Wireless Sensor Networks: Parameter Estimation, Performance Benchmarks, and Protocols*, Cambridge Univ. Press, 2009.
[15] P. Misra *et al.*, poster abstract, "Cleanhands: An Integrated Monitoring System for Control of Hospital Acquired Infections," *Proc. 14th Int'l. Conf. Information Processing in Sensor Networks*, New York, NY, 2015, pp. 348–49.

Biographies

Sabarish Sridhar is currently pursuing a B.Tech. in electronics and communications at M.S. Ramaiah Institute of Technology, Bangalore. He is passionate about healthcare, sensor networks, and smart systems. His current research interests include automating analog design using geometric programming, image processing with field programmable gate arrays, and development of embedded frameworks for sensor systems that are economically viable.

Prasant Misra [SM] is a senior MTS at the Robert Bosch Centre for Cyber Physical Systems in the Indian Institute of Science, Bangalore. He received his Ph.D. from the University of New South Wales, Sydney, Australia, in 2012. His current research interests include low-power sensing/communication, signal processing, and energy-efficient computing with a focus on system design and implementation within the general framework of cyber physical systems and the Internet of Things. He has many years of experience in technology development, and has worked in different roles and capabilities for Keane Inc. (now a unit of NTT Data Corporation), India; CSIRO ICT Centre, Australia; Red Lotus Technologies, United States; and SICS Swedish ICT, Sweden, the outcomes of which have resulted in either commercial products or publications in premier sensornet forums such as *ACM/IEEE IPSN* and *ACM TOSN*. His professional and research contributions have been recognized by numerous awards, of which it is noteworthy to mention the ERCIM Alain Bensoussan/Marie Curie Fellowship (2012) and the AusAID Australia Awards Leadership Program (2008). He has also served on the organizing/technical committees of a number of international conferences. He is a member of ACM, Secretary of the IEEE Computer Society (Bangalore Section), and an Associate Technical Editor of *IEEE Communications Magazine*.

Gurinder Singh Gill did not have a biography available at the time of publication.

Jay Warrior has over 20 years of creating new high-technology-based business opportunities for Agilent Technologies, Hewlett-Packard, Emerson, Fisher-Rosemount, and Honeywell in the United States and Asia. He is driven by a systems-wide perspective to problems, integrating long-term trends with strategy, technology development, and user-centric design techniques in his work. He puts these into practice at Mobiatrics LLC, a healthcare focused startup. He also works extensively with TiE and IESA on developing the IoT ecosystem for India. His previous experiences include his roles as chief technologist at the Robert Bosch Centre for Cyber Physical Systems at IISc, Bangalore, a technology transfer organization, chief technologist and strategist for the Network Solutions business at Agilent Technologies, and managing director of New Business Creation at Agilent. He has led the the development of multiple large-scale distributed systems, including the HART protocol, a de facto standard in process automation. He holds a Ph.D. in control and dynamical systems, and currently has over 25 patents covering key inventions in networking and diagnostics technology.